# APPLICATION

# FOR

# UNITED STATES LETTERS PATENT



TITLE: PROCESSING DATA PACKETS

APPLICANT: GILBERT WOLRICH, MARK B. ROSENBLUTH AND DEBRA BERNSTEIN

CERTIFICATE OF MAILING BY EXPRESS MAIL

Express Mail Label No. EL870691565US

I hereby certify that this correspondence is being deposited with the United States Postal Service as Express Mail Post Office to Addressee with sufficient postage on the date indicated below and is addressed to the Commissioner for Patents, Washington, D.C. 20231.

February 1, 2002
Date of Deposit


Signature

Gabriel Lewis
Typed or Printed Name of Person Signing Certificate

# PROCESSING DATA PACKETS

## BACKGROUND

This invention relates to processing of network packets.

Current store and forward network devices such as routers and switches are expected to be capable of processing data packets at increased line speeds of 10 Gigabits and higher. For minimum sized packets, the network device should be able to store newly received data packets to a memory structure at a rate at least equal to the arrival time of the packets. In addition, in order to maintain system throughput without dropping data packets, packets should be removed from the memory and transmitted at the packet arrival rate.

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a network system.

FIG. 2 is a block diagram of a network processor.

FIG. 3 is a block diagram of a cache data structure that illustrates enqueue and dequeue operations.

FIGS. 4A and 4B illustrate an enqueue operation.

FIGS. 5A and 5B illustrate a dequeue operation.

## DETAILED DESCRIPTION

Referring to FIG. 1, a network system 10 for processing data packets includes a source of data packets 12 coupled to an input of a network device 14. An output of the network device is coupled to a destination of data packets 16. The network device 14 can include a network processor 18 with memory data structures configured to store and forward the data packets to a specified destination. Such a network device 14 can include a network switch, a network router, or other network device. The source of data packets 12 can include other network devices 14 connected over a communications path operating at high data packet transfer line speeds such as an optical carrier (OC)-192 line, 10 Gigabit line, or other line speeds. Likewise, the destination of data packets 16 can include a similar network connection.

Referring to FIG. 2, the network processor 18 includes a processor 26 coupled to a first memory 30, a second memory 32, and a third memory 17. The third memory 17 has software instructions for managing the operation of the network processor 18. However, the instructions also can be stored in the first memory 30, the second memory 32, or a combination of the two memories. The processor 26 includes a queue manager 48 containing software components configured to manage a cache of queue descriptors ("the cache") having a tag portion 44a and a

data store portion 44b. The tag portion 44a of the cache 44 resides in the processor 26, whereas the data store portion 44b of the cache resides in the first memory 30. The tag portion 44a is managed by a content addressable memory (CAM) module 29 which can include hardware components configured to implement a cache entry replacement policy such as a least recently used (LRU) policy.

The data store portion 44b maintains a certain number of the most recently used (MRU) queue descriptors 46, each of which includes pointers 49 to a corresponding MRU queue of buffer descriptors 48, hereinafter, referred to as a MRU queue of buffers 48. In one implementation, the maximum number of MRU queue descriptors 46 in the data store portion 44b is sixteen. Each MRU queue descriptor 46 is referenced by a set of pointers 45 residing in the tag portion 44a. In addition, each MRU queue descriptor 46 is associated with a unique identifier so that it can be identified easily. Each MRU queue of buffers 48 stores pointers 53 to data buffers 38 residing in the second memory 32. In another embodiment, each MRU queue of buffers 48 can refer to the data buffers 38 by mapping to the data buffers. In one implementation, each queue of buffers 48 can be a linked list of buffers. Each data buffer 38 can store multiple data packets as provided from a receive buffer 20.

Least recently used (LRU) queue descriptors 50, which are not currently referenced by the data store portion 44b, reside in the first memory 30. Each LRU queue descriptor 50 also is associated with a unique identifier and includes pointers 51 to a corresponding LRU queue of buffers 52. Each LRU queue of buffers 52 contains pointers 57 to the data buffers 38 residing in the second memory 32.

The first memory 30 can include fast access memory such as static random access memory (SRAM). The processor 26 is coupled to the first memory 30 that has a space for storing data and memory controller logic 34 to handle queue commands and exchange data with the processor. The second memory 32 is coupled to the processor 26 as well as other components of the network processor 18. The second memory 32 can include dynamic random access memory (DRAM) that generally has a slower access time than SRAM but may provide fast burst mode access to the data buffers 38. Alternatively, the first memory 30 and/or the second memory 32 can reside externally to the network processor 18.

The receive buffer 20 is coupled to the processor 26 through a receive pipeline 22 and is configured to buffer data packets received from the source of data packets 12 (see FIG. 1). Each data packet can contain a real data portion, a source data portion representing the network address of the source of

the data and a destination data portion representing the destination network address where the data packet is to be delivered.

The receive pipeline 22 can include multiple processors using pipelining and multi-threading techniques to process the data packets. The receive pipeline 22 processes the data packets from the receive buffer 20 and stores the data packets in a data buffer 38. In addition, the receive pipeline 22 determines which queue is associated with the data buffer 38. The CAM module 29 is responsible for determining whether the data buffer 38 is associated with a MRU queue 48 and a corresponding MRU queue descriptor 46 or with a LRU queue 52 and a corresponding LRU queue descriptor 50.

Once the data packets are processed, the receive pipeline 22 generates enqueue requests 23 directed to the processor 26. Each enqueue request 23 represents a request to append a newly received buffer to the last buffer in the queue 48. The receive pipeline 22 can buffer a certain number of packets before generating each enqueue request 23. Consequently, the number of enqueue requests 23 are reduced because enqueue requests are not generated until a specified number of packets have arrived. An enqueue request 23 includes an address pointing to a queue descriptor of the queue of buffers 48a which further refers to a data buffer 38 associated with the received data packets. In

addition, each enqueue request 23 includes an identifier specifying either a LRU queue descriptor 50 or a MRU queue descriptor 46 associated with the data buffer 38.

The processor 26 processes each enqueue request 23 and, in response, generates an enqueue command 13 directed to the memory controller 34. The enqueue command 13 may include a queue identifier specifying a MRU queue descriptor 46 residing in the data store portion 44b. The MRU queue descriptor 46 includes a pointer 49 to a corresponding queue 48. The queue 48 is updated to point to the data buffer 38 that has the received data packet. In addition, the MRU queue descriptor 46 is updated to reflect the updated state of the MRU queue 48. The MRU queue descriptor 46 can be updated in a fast and efficient manner because the queue descriptor already is in the data store portion 44b.

On the other hand, the enqueue command 13 may include a queue identifier specifying a LRU queue descriptor 50 that points 51 to a LRU queue 52. In that case, the queue manager 27 replaces a particular MRU queue descriptor 46 with the LRU queue descriptor 50. As a result, the LRU queue descriptor 50 and the corresponding LRU queue 52 are referenced by the data store portion 44b. In addition, the newly referenced LRU queue 52 associated with the LRU queue descriptor 50 is updated to point to the data buffer 38 containing the received data packet.

A transmit scheduler 24 is coupled to the processor 26 and is responsible for generating dequeue requests 25 based on criteria such as when the number of buffers in a particular queue reaches a predetermined level. Each dequeue request 25 represents a request to remove the first buffer from the queue 48. The transmit scheduler 24 also may include scheduling algorithms for generating dequeue requests 25 such as "round robin", priority based, or other scheduling algorithms. The transmit scheduler 24 also can be configured to use congestion avoidance techniques such as random early detection (RED) which involves calculating statistics for the packet traffic. The processor 26 generates dequeue commands 15 directed to the memory controller 34 in response to receiving dequeue requests 25. The processor 26 can include a programming engine having a multiple context architecture such that each context can be assigned a queue of buffers. In one implementation, the processor 26 has a ring data structure for handling enqueue requests 23 and dequeue requests 25.

Similar to the enqueue command discussed above, each dequeue command 15 may include a queue identifier specifying a queue descriptor such as a MRU queue descriptor 46 that points to a MRU queue 48. As discussed above, the MRU queue 48 includes pointers 53 to data buffers 38. The data in the buffers 38 referenced by the pointer 53 is returned to the

processor 26 for further processing. As a result, the queue 48 is updated and no longer points to the returned data buffer 38 since it is no longer referenced by the data store portion 44b.

On the other hand, the dequeue command 15 may include a LRU queue descriptor 50 that points to a LRU queue 52. In that case, which is similar to the queue command 13 discussed above, the queue manager 27 replaces a particular MRU queue descriptor with the LRU queue descriptor. This includes performing a "write back" operation in which the replaced queue descriptor is written back to the first memory 30. As a result, the replacement MRU queue descriptor 46 and the corresponding MRU queue buffer 48 are referenced by the data store portion 44b. The data buffer 38 that was pointed to by the queue 48 is returned to the processor 26 for further processing. As a result, the queue buffer 48 is updated and no longer points to the data buffer 38 since it is no longer referenced by the data store portion 44b.

The processor 26 is coupled to a transmit pipeline 28 which is responsible for handling the data buffers 38 for transmission to a transmit buffer 36. The transmit pipeline 28 may include multi-threading and pipelining techniques to process the data buffers. For example, each thread may be assigned to a particular queue of buffers, thereby allowing multiple queues to be processed in parallel.

The transmit buffer 36 is coupled to the transmit pipeline 28 and is responsible for buffering the data buffers 38 received from the transmit pipeline.

Referring to FIG. 3, the operation of the cache 44 is shown. The tag portion 44a can include 16 entries containing pointers 45 to a corresponding queue descriptor 46 in the data store portion 44b. For purposes of illustration only, the following discussion focuses on the first entry 1 in the tag portion 44a. The first entry 1 is associated with a pointer 45a pointing to a MRU queue descriptor 46a residing in the data store portion 44b. The queue descriptor 46a is associated with a MRU queue 48a. Each buffer, such as a first buffer A, includes a pointer 53a to a respective data buffer 38a in the second memory 32. In addition, each buffer, includes a buffer pointer 55a pointing to a subsequent ordered buffer B. It should be noted that buffer pointer 55c associated with a last buffer C has a value set to NULL to indicate that it is the last buffer in the queue 48a.

The queue descriptor 46a includes a head pointer 49a pointing to the first buffer A and a tail pointer 49b pointing to the last buffer C. An optional count field 49c maintains the number of buffers in the queue 48a. In the illustrated example, the count field 49c is set to the value 3 representing the buffers A to C. As discussed in further detail below, the head

pointer 49a, the tail pointer 49b and the count field 49c may be modified in response to enqueue requests 23 and dequeue requests 25.

As indicated by FIGS. 4A and 4B, in response to receiving an enqueue request 23, the processor 26 generates 100 an enqueue command 13 directed to the memory controller 34. In this example, the enqueue request 23 is associated with a subsequent data buffer 38d received after data buffer 38c. The enqueue request 23 includes an identifier specifying the queue descriptor 46a and an address associated with the data buffer 38d residing in the second memory 32. The tail pointer 49b in the queue descriptor 48a currently pointing to buffer C is returned to the processor 26. As discussed below with reference to block 110, each enqueue request 23 is evaluated to determine whether the queue descriptor associated with the enqueue request is currently in the data store portion 44b. If it is not, then a replacement function is performed.

The buffer pointer 55c associated with buffer C currently contains a NULL value (FIG. 3) indicating that it is the last buffer in the queue 48a. The buffer pointer 55c is set 102 to point to the subsequent buffer D (FIG. 4B). This is accomplished by setting the buffer pointer 55c to the address of the buffer D.

Once the buffer pointer 55c has been set (block 102), the tail pointer 49b is set 104 to point to buffer D as shown by dashed line 61. This also may be accomplished by setting the tail pointer to the address of the buffer D. Since buffer D is now the last buffer in the queue of buffers 48a, the value of the buffer pointer 55d is set to a NULL value. Moreover, the value in the count field 49c is updated to four reflecting the buffers A to D in the queue 48a. As a result, the buffer D has been added to the queue 48a by using the queue descriptor 46a residing in the data store portion 44b.

The processor can receive 106 a subsequent enqueue request 23 associated with the same queue descriptor 46a and queue 48a. Using the example above, it is assumed that the processor 26 receives the enqueue request 23 in connection with a newly arrived data buffer 38e. It also is assumed that the data buffer 38e is associated with the queue descriptor 46a. In that case, the tail pointer 49b can be set 108 to point to buffer E which is represented by replacing the dashed line 61 pointing to buffer D with the dashed line 62 pointing to buffer E (FIG. 4B). The tail pointer 49b is updated without having to retrieve it because it is already in the data store portion 44b. As a result, the latency of back-to-back enqueue operations to the same queue of buffers can be reduced.

Suppose, however, that the queue descriptor 46a currently occupying the first entry 1 in the data store portion 44b is not associated with the newly arrived data buffer 38e. In that case, the processor performs 110 a replacement function, which includes removing one of the queue descriptors from the data store portion 44b. For example, the replacement policy can be based on a LRU policy in which a queue descriptor that has not been accessed during a predetermined time period is removed from the data store portion 44b. The removed queue descriptor is written back to the first memory 30 (FIG. 2) and is replaced in the data store portion 44b with the queue descriptor associated with data buffer 38e. Once the replacement function has been completed, queue operations associated with the enqueue request are performed as discussed above.

As indicated by FIGS. 5A and 5B, in response to receiving 200 a dequeue request 25, the processor 26 generates 200 a dequeue 15 command directed to the memory controller 34. In this example, the dequeue request 25 is associated with the data buffer 38a residing in the second memory 32. The dequeue request 25 represents a request to retrieve the data buffer 38a from the second memory 32. Once the data buffer 38a is retrieved, it can be transmitted from the second memory 32 to the transmit buffer 36. The dequeue request 25 includes an identifier specifying the queue descriptor 46a and an address

associated with the data buffer 38a residing in the second memory 32. The head pointer 49a of the queue descriptor 46a points to the first buffer A which in turn points to data buffer 38a. As a result, the data buffer 38a is returned to the processor 26.

The head pointer 49a is set 202 to point to the next buffer B in the queue 48a as shown by dashed line 64. This is accomplished by setting the head pointer 49a to the address of buffer B. The value in the count field 49c is updated to four reflecting the remaining buffers B to E. Thus, the data buffer 38a is retrieved from the queue 48a using the queue descriptor 46a residing in the data store portion 44b.

The processor 26 can receive 204 subsequent dequeue requests associated with the same queue buffer 48a and the queue descriptor 46a. Continuing the example above, it is assumed that the processor 26 receives a further dequeue request associated with the data buffer B. As discussed above, the head pointer 46a currently points to buffer B (represented by the dashed line 64 in FIG. 5B) which is the first buffer since the reference to buffer A was removed. It also is assumed that the data buffer B also is associated with queue descriptor 46a. In that case, the head pointer 49a can be set 206 to point to buffer C, as shown by a dashed line 65 (FIG. 5B), without having to retrieve the head pointer 49a because it is already in the

data store portion 44b. As a result, the latency of back-to-back dequeue operations to the same queue can be reduced.

Suppose, however, that the queue descriptor 46a currently occupying the first entry 1 in the data store portion 44b and the queue descriptor are not associated with the data buffer 38b. In that case, the processor performs 208 a replacement function similar to the one discussed above. Once the replacement function has been completed, operations associated with the dequeue request are performed as previously discussed above.

The cache of queue descriptors 44 can be implemented in a distributed manner such that the tag portion 44a resides in the processor 26 and a data store portion 44b resides in the first memory 30. As a result, data buffers 38 that are received from the receive buffer 20 can be processed quickly. For example, the second of a pair of dequeue commands can be started once the head pointer for that queue descriptor is updated as a result of the first dequeue memory read of the head pointer. Similarly, the second of a pair of enqueue commands can be started once the tail pointer for that queue descriptor is updated as a result of the first enqueue memory read of the tail pointer. In addition, using a queue of buffers, such as a linked list of buffers, allows for a flexible approach to processing a large number of

queues. Data buffers can be quickly placed on or removed from the queue of buffers.

Various features of the system can be implemented in hardware, software, or a combination of hardware and software. For example, some aspects of the system can be implemented in computer programs executing on programmable computers. Each program can be implemented in a high level procedural or object-oriented programming language to communicate with a computer system.

Furthermore, each such computer program can be stored on a storage medium, such as read-only-memory (ROM), readable by a general or special purpose programmable computer, for configuring and operating the computer when the storage medium is read by the computer to perform the functions described above.

Other implementations are within the scope of the following claims.